Vecstra: An Efficient and Scalable Geo-spatial In-Memory Cache

Yiwen Wang[*]
supervised by Marcos Antonio Vaz Salles
Database Management System Group
University of Copenhagen, Denmark
y.wang@di.ku.dk

ABSTRACT

Nowadays a huge amount of geospatial data are generated and widely used in a variety of areas. In this way, there is an extensive range of database systems that support geospatial services but fall behind in coping with increasingly demanding high throughput, low latency and efficient resource usage requirements. Meanwhile, in-memory databases have become more prevalence due to increasing capability as well as decreasing price of the memory. To take advantage of the fast I/O speed and overhead reduction of memory-based storage, in-memory geospatial data management systems are created to overcome drawbacks of disk-based geospatial databases. However, to the best of our knowledge, current in-memory caches are not fully-featured and do not completely utilize the benefits of widely used geospatial standards. To fill this gap, we specialize to the API of standard geospatial services and seek to achieve by this method a much higher performing in-memory geospatial cache implementation than possible by combining generic components such as a geospatial application server and SQL database. To that end, an efficient and scalable OGC standard-compliant in-memory geospatial cache *Vecstra* is built. Furthermore, we conduct experiments on Vecstra and analyze the preliminary results to formulate research opportunities.

1. INTRODUCTION

There has a considerable increasing of geospatial data usage in today's society in many areas, such as precision agriculture [22] and social media streams querying [38]. Meanwhile, an explosion of the Internet of Things (IoT) has recently helped increase the generation of geospatial data [20]. In the agriculture field, there is a huge amount of data from satellites and sensors that can provide massive geospatial information about terrain and crop status. By integrating a range of data and providing the information, that enables farmers to tailor their management according to the local and actual conditions in the field. Since the importance and wide societal relevance of spatial data, many platforms provide geospatial services, necessitating specialized data management solutions. GeoServer [12], for example, is the backend of a number of such services and builds its geospatial database on PostGIS [17].

On the other hand, due to the increasing capability and the decreasing price of memory, there is higher possible to cache substantial amounts of geospatial data in-memory. Storing data in memory is especially valuable for running concurrent workloads as it eliminates the inevitable bottlenecks caused by disk-centric architectures [21]. To take advantage of the benefits of object caching systems, there are widely used web services caches such as Memcached [15] and Redis [9]. Now more and more classic databases providers have developed their in-memory features and systems. For instance, in-memory database TimesTen [27] owned by Oracle and memory-optimized OLTP engine Hekaton [13] in SQL Server. Moreover, multi-core servers also help to achieve performance benefits and can yield significant gains. Multi-core processors can simultaneously handle multiple requests that can help reduce latency when amounts of concurrent requests occur, which is a very likely situation with skew in geospatial services [1].

Despite the popularity of geospatial databases such as PostGIS, however, their design has benefited little from recent developments in in-memory databases. PostGIS still utilizes disk as its primary storage, which raises the problem to reduce expensive geospatial data querying cost due to disk I/O and system overheads. Since geospatial services are experiencing growing popularity, this problem is made worse by large numbers of concurrent requests. In-memory database techniques and multi-core hardware hold promise to offer a solution to this problem, while at the same time increasing throughput and resource utilization.

Existing Systems. Currently, there are many geospatial databases that provide services to clients, as shown in Figure 1. There are disk storage databases for batch processing and multiple types of spatial queries such as SpatialHadoop [14] and Hadoop-GIS [2]. With the availability of larger and cheaper main memory, to take the advantage of that, data management systems such as MemSQL [35] are built based on main memory to improve response times. MemSQL is a in-memory database that stores data and allows for quickly analyzing and processing data. And there are transactions for the small interactive requests, to support such requests, a number of specialized systems have been

[*]Work performed in the context of the Future Cropping partnership, supported by Innovation Fund Denmark.

Proceedings of the VLDB 2018 Ph.D. Workshop, August 27, 2018. Rio de Janeiro, Brazil.

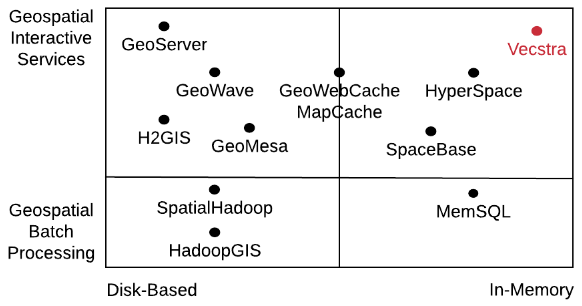

Figure 1: Related Work Summary.

developed. For example, GeoServer that based on PostGIS we discussed above, and GeoMesa [19], a spatio-temporal distributed database, also for GeoWave [3] and H2GIS [6]. There are also disk-based caches such as GeoWebCache [28] integrated with GeoServer that can store different types of map. MapCache [7] is also a disk cache that similar with GeoWebCache. As we can see, even though many such geospatial systems exist, only a few are based on main memory. To the best of our knowledge, two systems stand out in this category. One is HyperSpace [32], a geospatial main-memory database system that can fastly process geospatial queries. The other one is SpaceBase [40], a system that allows handling concurrent geospatial queries with low latency based on R-Tree indexing. But those two described in-memory caches fail to support widespread geospatial data such as raster data. Also, these caches also have limited support spatial data structures, which can substantially affect processing efficiency in main memory. Meanwhile, latency and efficiency in resource usage of modern multi-core machines are not discussed in these caches. For the temporally skewed geospatial query, keep scalability of systems when large numbers of users and request volumes happen which is a highly possible scenario should be taken into consideration as well.
Contribution. To fill the gap as described above, inspired by the criticism of "One size fits all" in [39], we specialize the design of our cache to the particular geospatial standards. We build OGC standards [8] compliant in-memory cache Vecstra that support spatial data structures to achieve high performance in geospatial services. There are many standards developed by OGC. Web Feature Service (WFS) [42] and Web Coverage Service (WCS) [41] are two of the most widely used standards. WFS is used to serve geographic feature information that mostly used in web-based client applications for GIS data editing. WCS is used to provide information on coverages. Known WFS/WCS implementations include open-source systems such as rasdaman [4], MapServer [26] and GeoServer. Commercial implementations include Envitia[1], Pyxis WorldView Studio[2], ESRI ArcGIS, among others. Most current disk-based geospatial systems implement these two standards; however, in-memory storage systems such as HyperSpace and SpaceBase fail to implement complete services. Given the popularity of these OGC standards, it would be ideal if a cache can provide support to them and that can help the cache directly integrated into existing disk-based geospatial systems. One approach to build the cache could be to build or extend existing SQL based systems such as HyPer or SpaceBase that provides features for SQL require for data. However, since providing a specialized interface, we have opportunities to provide a much better implementation. Targeted advantages of Vecstra are:

1. Use of tuned in-memory data structures for geospatial data, leading to lower latency;
2. Thread management that is optimized for the interface of a geospatial cache supporting operations over geospatial layers, leading to lower overhead and better scalability;
3. Open-source system that is standards-compliant, offering integrated WFS and WCS interfaces. And in the future, it is possible to extend Vectra with standard WPS as well.

The remainder of this paper is organized as follows. In Section 2 we illustrate our solution to build Vecstra and present preliminary experiment results. In Section 3, based on these results, we discuss research challenges and conclude in Section 4.

2. SOLUTION

To explore a solution to the problems in Section 1, we build a scalable, fully functional in-memory geospatial cache called *Vecstra* that can provide high efficient services and low latency response to geospatial data requests, operating on a multi-core computer for the interest of high performance. This section focus on how we build Vecstra, including API design, architecture, and experiments.

2.1 API design

Vesctra is an in-memory storage cache that provides services to clients to obtain geospatial data for calculations and analysis goals. Vecstra implements the standard WFS and WCS interfaces, which we analyzed to collect the functions that Vecstra should support. Abiding by these APIs is advantageous as this allow us to specialize the whole system to particular operations as opposed to providing a generic implementation.

Every API method in Vecstra is invoked by user requests to an instance v of Vecstra and the design of them are as follows. The *v.wfs/wcs::Add(l)* and *v.wfs/wcs::Remove(l)* add and remove vector or raster layers in v, respectively. The method *v.wfs/wcs::GetCapabilities* provides basic information of all vector or raster layers and services in v, while *v.wfs::GetFeature(l, bbox)* and *v.wcs::GetCoverage(l, bbox)*, respectively, provide full information in the intersection area of vector or raster layers l and bounding box *bbox* in v. Finally, *v.wfs::DescribeFeatureType(l)* and *v.wcs::DescribeCoverage(l)* obtain description and geoinformation of vector or raster layers in v.

The API opens up opportunities for specialization of the implementation. For example, updates are always at layer granularities, but queries typically read small parts of a few layers that are intersecting with a window in space. We observe that services often rely on PostGIS, which allows them to satisfy linearizability, so to keep consistent semantics we end up with the requirement to maintain linearizability for concurrent operations in Vecstra. As we will see in the next section, we can exploit these characteristics in the design of Vecstra to support the API above more efficiently than alternative generic designs based on an SQL geospatial database.

[1]https://www.envitia.com/
[2]https://www.pyxisglobe.com/

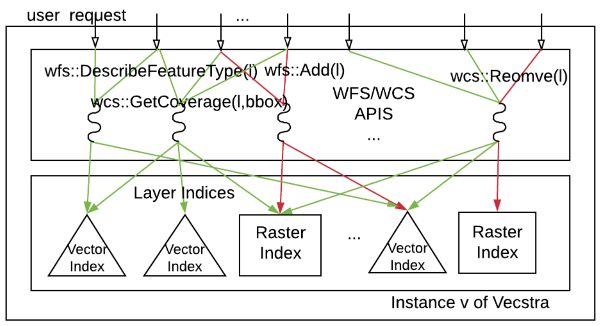

Figure 2: Architecture of Vecstra.

2.2 Architecture

Figure 2 illustrates the architecture of Vecstra. Vecstra contains both vector and raster data which are two primary types of spatial data in GIS that most commonly used in geospatial services [10]. Based on the results in [37], we choose R-Tree indices for the spatial intersection of vector data and currently we use multi-dimensional array indices for raster data. Essentially, query windows are for a number of layers and initially update operations are coarse-grained at layer granularities. Because the query operations usually do not read all layers and also in order to support the layer-level update operations, the design of indices is partitioned by layers, i.e., every index corresponds to one layer.

To exploit multi-core benefits, multi-threading is used to gain performance. Concurrent request threads are assigned to different cores. The multiple threads running in Vecstra are executing both queries and updates. Index structures such as the high-concurrency locking R-Tree proposed by [25] can be exploited to utilize concurrent threads. However, as shown in Section 2.1, modification requests in Vecstra are coarse-grained at layer granularities, so fine-grained concurrency in indices has been unnecessary so far. We can explore other possibilities where several particular layers are combined into one single index as future work. Currently, we utilize a concurrent hash map [33] to store the indices of layers, which supports full concurrency of query operations and high expected concurrency for update operations. In addition, to minimize synchronization costs, a thread performs indexing on a given layer without synchronization, and then uses the concurrent hash map to synchronously store the built index in a single operation. Based on the OGC standards and to integrate with the current systems such as GeoServer, there is an HTTP communication interface that is provided to users. Users provide GET requests to obtain WCS and WFS services from Vecstra.

2.3 Preliminary Results

Prototype and Workload. Vecstra prototype implement architecture of Figure 2 based on C++11 and is compiled by gcc 5.4.0. Realistic data are used for experiments. Raster data we used are Field polygons, Soil, Rain distribution layers and a topographic layer of Denmark, DTK/Kort25, for raster data. Used workload designed based on precision farming scenario [21]. Queries are generated for fields and update requests are for layers. Currently, all queries and requests are uniform random generated but in the future, we can design a skew request mechanism based on the real request log. We conduct our experiments on an AWS EC2 r4.4xlarge instance of Ubuntu server 16.04 LTS.

Results. At presesent, we use single read-write lock to achieve automicity of Vecstra which is expected to be not scalable. Therefore, we do our experiment only focus on queries. We analysis the time in server side to show which parts the latency located that can be improved in future work. Now for vector data, most part of latency is located in geometry intersection (around 50%-64%) and for raster data, the highest portion of latency is located in data extraction (around 89%-95%).

3. RESEARCH CHALLENGES

The work presented so far was done in the first 9 months as a Ph.D. candidate. Based on the preliminary results from the evaluation of Vecstra, we describe the future research directions to overcome shortcomings described in Section 3.

Optimization for Geospatial Window Queries. A geospatial range query is a very common database operation that retrieves geospatial information located between an upper and lower boundary[30]. The refine step in geometric intersection time is the biggest part of WFS range query latency and data extraction use the majority of WCS range query processing time, that can be seen from the server side evaluation results. Spatial indexing is used to support spatial selection [16]. Now, the minimum bounding rectangle is used as the geometric key of our vector data and the most time of vector range query is located in intersection calculation, [18] implemented non-leaf and leaf MBR optimizations techniques in Oracle Spatial. A new optimization approach adds on top of this technique to reduce the intersection time is one of our goals.

Now our raster data is continuously and linearly stored in an array from left to right and from top to bottom. We calculate the intersected cells by decomposing the range query to a set of range sections, then extracting data block-by-block. Now we can see the latency of processing of raster data extract is relatively high. In order to make this process faster, a better data partitioning method should be used[31]. Now Q-Tree and R-Tree are two most common indexes in commercial database systems [5]. Due to most geo range queries are rectangle shape, we consider exploring a better range indexing for raster data. [29] use space-filling curves to index the data, but this proposed curve-design method is heuristic. Therefore, an efficient curve-exploration approach and valuating the best combination of curves in an in-memory cache are challenges that we will pursue as an avenue for future work.

Efficient Updates in Multi-Core Cache. Concurrency control should be used to manage requests to ensure the semantics of Vecstra. However, the current scalability of our shared everything implementation is still limited by potential synchronization bottlenecks in the presence of update skew towards certain layers of geospatial data [23]. A composable thread management approach [24] will be explored in future work. Also, in the future, we can utilize lock-free mechanisms to tackle this problem. Unlike work in conventional updates to spatial data structures [36], this mechanism will exploit the concurrent updates of Vecstra and recent work on [11] for the maximal performance.

Design of Integration Language for Geospatial Web services. Now Vecstra provides WCS and WFS to users; however, queries are only for raster or vector data, which is the same with the current geospatial service databases.

We want to go further in this aspect, that is, to design an integration language to combine geospatial data. For example, to stay standards-compliant, we can implement Web Processing Service (WPS) [34] that can help users define the execution way of a process and also handle the output of the process. Widely used standard Web Map Service (WMS) [43] is also a solution. By doing that, a user can get geospatial information such as both raster and vector data from one single query which can provide them a better view in a convenient way. Further research can be done on the basis of this to achieve a better balance expressivity and performance; for instance, we can provide specialized support for a subset of scripts such as support adapting data to scale in zoomable maps.

4. CONCLUSION

In this paper, we investigated current geospatial data management systems and discussed challenges. To address these challenges, our goal is to build an efficient and scalable OGC standard-compliant in-memory geospatial cache. We specialized API of standard geospatial services and sought to achieve by this method a much higher performing in-memory geospatial cache implementation than possible by combining generic components such as a geospatial application server and SQL database. To achieve that, we started by building our initial version of Vecstra and obtained preliminary results on it. We described the problems that need to be solved in our future research and proposed possible methods. First, find an optimized window query approach. Second, exploit the efficient updates in the multi-core cache. Third, design an integration language for geospatial services.

5. REFERENCES

[1] J. Ahmed, M. Y. Siyal, S. Najam, and Z. Najam. Challenges and Issues in Modern Computer Architectures. In *Fuzzy Logic Based Power-Efficient Real-Time Multi-Core System.* Springer, 2017.

[2] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. H. Saltz. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *Proc. VLDB*, 6 11, 2013.

[3] A. Annex. Geowave provides geospatial and temporal indexing on top of Accumulo and HBase, 2018.

[4] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann. The multidimensional database system RasDaMan. In *Acm Sigmod Record*, volume 27, 1998.

[5] N. Black. Geo-location custom indexes, 2018. US Patent 9,875,321.

[6] E. Bocher, G. Petit, N. Fortin, and S. Palominos. H2gis a spatial database to feed urban climate issues. In *ICUC9*, 2015.

[7] T. Bonfort and T. Bonfort. MapCache: The Fast Tiling Server From The MapServer Project. 2013.

[8] M. Botts, G. Percivall, C. Reed, and J. Davidson. Ogc® sensor web enablement: Overview and high level architecture. In *GeoSensor networks.* 2008.

[9] J. L. Carlson. *Redis in action.* Manning Publications Co., 2013.

[10] K.-T. Chang. *Introduction to geographic information systems.* McGraw-Hill Science/Engineering/Math, 2015.

[11] A. T. Clements, M. F. Kaashoek, N. Zeldovich, R. T. Morris, and E. Kohler. The scalable commutativity rule: Designing scalable software for multicore processors. *ACM TOCS*, 32, 2015.

[12] J. Deoliveira. Geoserver: uniting the GeoWeb and spatial data infrastructures. In *Proc. GSDI-10*, 2008.

[13] C. Diaconu, C. Freedman, E. Ismert, P.-A. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling. Hekaton: SQL server's memory-optimized OLTP engine. In *Proc. 2013 ACM SIGMOD.*

[14] A. Eldawy and M. F. Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *ICDE, IEEE*, 2015.

[15] B. Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004, 2004.

[16] R. H. Güting. An introduction to spatial database systems. *VLDB*, 3, 1994.

[17] S. Holl and H. Plum. Postgis. *GeoInformatics*, 3, 2009.

[18] Y. Hu, S. Ravada, R. Anderson, and B. Bamba. Topological relationship query processing for complex regions in Oracle Spatial. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems.* ACM, 2012.

[19] J. N. Hughes, A. Annex, C. N. Eichelberger, A. Fox, A. Hulbert, and M. Ronquest. Geomesa: a distributed architecture for spatio-temporal fusion. In *SPIE: Geospatial Informatics, Fusion, and Motion Video Analytics V*, volume 9473, 2015.

[20] A. P. Iyer and I. Stoica. A scalable distributed spatial index for the internet-of-things. In *Proc. ACM SoCC*, 2017.

[21] M. E. Jacobsen. A Scalable Geospatial System for Range Queries in Data-Driven Precision Agriculture. Master's thesis, University of Copenhagen, Denmark, 2017.

[22] J. H. Jeppesen, E. Ebeid, R. H. Jacobsen, and T. S. Toftegaard. Open geospatial infrastructure for data management and analytics in interdisciplinary research. *Computers and Electronics in Agriculture*, 145:130–141, 2018.

[23] J. H. Jeppesen, R. H. Jacobsen, R. N. Jørgensen, and T. S. Toftegaard. Towards data-driven precision agriculture using open data and open source software. In *International Conference on Agricultural Engineering*, 2016.

[24] R. Johnson, I. Pandis, and A. Ailamaki. Eliminating unscalable communication in transaction processing. *The VLDB Journal*, 23, 2014.

[25] M. Kornacker and D. Banks. High-concurrency locking in R-trees. In *VLDB*, volume 95, 1995.

[26] B. Kropla. *Beginning MapServer: open source GIS development.* Apress, 2006.

[27] T. Lahiri, M.-A. Neimat, and S. Folkman. Oracle TimesTen: An In-Memory Database for Enterprise Applications. *IEEE Data Eng. Bull.*, 36, 2013.

[28] Y.-F. Nie, H.-L. Liu, and H. Xu. Research on GeoWebCache tile map service middleware. *Science of Surveying and Mapping*, 36, 2011.

[29] S. Nishimura and H. Yokota. Quilts: Multidimensional Data Partitioning Framework Based on Query-Aware and Skew-Tolerant Space-Filling Curves. In *Proc. ACM SIGMOD*, 2017.

[30] M. Orru, R. Paolillo, A. Detti, G. Rossi, and N. B. Melazzi. Demonstration of OpenGeoBase: The ICN NoSQL spatio-temporal database. In *IEEE LANMAN*, 2017.

[31] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance in spatial data structures. In *Proc. PODS*, 1993.

[32] V. Pandey, A. Kipf, D. Vorona, T. Mühlbauer, T. Neumann, and A. Kemper. High-performance geospatial analytics in hyperspace. In *Proc. ACM SIGMOD*, 2016.

[33] J. Reinders. *Intel threading building blocks: outfitting C++ for multi-core processor parallelism.* " O'Reilly Media, Inc.", 2007.

[34] P. Schut and A. Whiteside. Opengis web processing service. *OGC project document*, 2007.

[35] N. Shamgunov. The MemSQL In-Memory Database System. In *IMDM@ VLDB*, 2014.

[36] D. Šidlauskas, S. Šaltenis, and C. S. Jensen. Processing of extreme moving-object update and query workloads in main memory. *The VLDB Journal*, 23, 2014.

[37] B. Sowell, M. V. Salles, T. Cao, A. Demers, and J. Gehrke. An experimental analysis of iterated spatial joins in main memory. *Proc. VLDB*, 6, 2013.

[38] A. Stefanidis, A. Crooks, and J. Radzikowski. Harvesting ambient geospatial information from social media feeds. *GeoJournal*, 78, 2013.

[39] M. Stonebraker and U. Cetintemel. " one size fits all": an idea whose time has come and gone. In *Proc. IEEE ICDE 2005.*

[40] T. P. Universe. Introducing Spacebase: a New Realtime Spatial Data Store, 21 March, 2012.

[41] O. WCS. Web Coverage Service, 2010.

[42] O. WFS. Web Feature Service. *OGC. URL: http://www. opengeospatial. org/standards/wfs*, 2005.

[43] O. WMS. Web Map Service, 2004.